UNIVERSIDAD CARLOS III DE MADRID
ESCUELA POLITÉCNICA SUPERIOR



INGENIERÍA TÉCNICA EN INFORMÁTICA DE GESTIÓN

TRABAJO DIRIGIDO

Acceso manual al juego AI-LIVE

Alumno: Iván Uzquiano Mateo
Tutor: Javier Ortiz Laguna
Año: 2008

Índice

1. Introducción

Este trabajo dirigido se basa en la creación de una interfaz manual que permita al usuario elegir las decisiones que va a tomar un jugador en el juego AI-LIVE. Éste fue desarrollado anteriormente como proyecto de fin de carrera por otros alumnos de la Universidad Carlos III de Madrid.

AI-LIVE es un juego similar a **Los Sims** que fue creado con el objetivo de simular el mundo real, donde los humanos son jugadores (denominados actores) que toman decisiones implementadas mediante técnicas de inteligencia artificial. Éstas se ordenan según las características propias de cada jugador, el cuál interactúa con su entorno (también llamado escenario) formado por objetos y otros jugadores.

El juego sigue el modelo cliente-servidor, de manera que el servidor es el encargado de asignar los turnos a cada uno de los actores para que puedan realizar las acciones pertinentes. Además, éste se encarga de actualizar el estado de los objetos y los actores según las acciones solicitadas.

Actualmente se está ampliando el juego con nuevos módulos que hacen que se asemeje más a la realidad, como son la implementación de la interfaz gráfica, la influencia del estado de ánimo en la toma de decisiones, los roles padre-hijo, etc.

Algunos módulos ya implementados son:

- Interacción con objetos del escenario actual: cama, nevera, ordenador, etc.
- Interacción con otros actores: hablar sobre determinados temas, otros actores, gustos, etc.
- Modificaciones de las características de cada actor (estado de ánimo, higiene, hambre, etc.) según la interacción con objetos y otros actores.
- Cambio de escenario para realizar otras acciones.

2. Objetivos

Las decisiones de los actores de AI-LIVE están controladas mediante técnicas de inteligencia artificial según las necesidades actuales de cada uno. Esto hace que los actores se comporten de manera automática, por lo que parece necesario crear una interfaz manual que permita a un usuario controlar a un actor.

Para ello, el usuario necesitará conocer las acciones que puede realizar el actor manual en el turno actual y, una vez mostradas, elegir la acción para enviarla al servidor y que éste la pueda ejecutar.

Esto permitirá simular el comportamiento humano en la toma de decisiones dentro del juego, y comprobar cómo influyen éstas en las acciones que realizan los actores implementados con inteligencia artificial.

Además, el estudio de las decisiones tomadas por el actor manual proporcionará información que permita la creación de futuros actores implementados con inteligencia artificial, cuyo comportamiento se asimile más al de un ser humano real.

3. Estudio

Antes de la realización del interfaz manual, es necesario realizar un estudio de cómo está estructurado el juego AI-LIVE, tanto física como lógicamente.

3.1. Jerarquía de archivos

Para proporcionar una visión física de cómo están estructurados los archivos, se muestra a continuación la jerarquía de directorios y archivos más significativos:

AI-LIVE/
- **client-manual/**
  - **profiles/**
    - *DEFAULT_ACTOR.profile*
  - *client*
  - *client.c*
  - *client.clp*
  - *client.ini*
  - *run*
  - *traza.txt*
  - *ontology.clp*
- **server/**
  - *initial.state*
  - *run*
  - *server*
  - *server.c*
  - *server.clp*
  - *server.ini*
  - *ontology.clp*

A continuación se va a describir el contenido de los archivos comunes al servidor y a los clientes en el juego:

- *ontology.clp*: contiene la definición de la jerarquía de clases del juego. Debe ser la misma tanto en el servidor como en todos los clientes.

- *run*: script que ejecuta las sentencias del archivo *.ini* perteneciente al directorio donde se encuentra.

Los demás archivos serán descritos en próximos apartados.

3.2. Arquitectura del juego

Actualmente el juego está formado por un servidor y uno o más clientes que se pueden conectar a éste mediante el establecimiento de un socket TCP/IP, utilizando un protocolo de comunicación especial creado para el juego. Se sigue una política rotatoria para controlar el turno de los clientes.

3.2.1. Servidor

El servidor es el encargado de ejecutar las acciones solicitadas por los actores en cada turno. Además envía la información del estado actual de todas las entidades del juego (objetos y actores) a los clientes, encargándose de actualizar ésta una vez ha sido modificado su estado.

Para saber cómo están dispuestos los objetos del escenario, el servidor utiliza el archivo *initial.state*, que contiene la definición de los objetos iniciales de las clases incluidas en la ontología.

El servidor está formado por dos archivos principales:

- *server.c*: se encarga de realizar las conexiones con los clientes, recibir las solicitudes de acciones de los actores, actualizar y enviar el estado de las entidades a los clientes, controlar los turnos de los actores y gestionar el motor de inteligencia artificial del servidor. Requiere de la ontología (*ontology.clp*), del estado inicial (*initial.state*) y del archivo *server.clp*.

- *server.clp*: contiene el sistema de producción implementado en **CLIPS** encargado de la creación de nuevos actores, del control de las decisiones de cada actor, y de la actualización del estado del juego mediante los métodos y funciones definidos.

3.2.2. Clientes

Los clientes son los encargados de tomar las decisiones y enviarlas al servidor para que las ejecute, así como de recibir el estado actual de las entidades del juego.

Una vez finalizadas sus ejecuciones, cada uno genera el archivo *traza.txt*, que contiene la secuencia de decisiones tomadas por el actor hasta el fin de su ejecución.

Cada cliente está formado por dos archivos principales:

- *client.c*: se encarga de conectarse al servidor, enviar la decisión del actor, recibir el estado actual y gestionar el motor de inteligencia artificial del cliente. Requiere de la ontología (*ontology.clp*, la misma que en el servidor) y del archivo *client.clp*.

- *client.clp*: contiene el sistema de producción implementado en **CLIPS** encargado de la toma de decisiones del actor, así como los métodos y funciones necesarios para la creación de la traza (*traza.txt*).

3.3. Estructura del archivo *client.c*

El archivo *client.c* es el principal objeto de estudio, ya que éste va a ser en donde se va a implementar la nueva funcionalidad encargada de alcanzar el objetivo del trabajo dirigido. Por ello se va a mostrar a continuación cómo está organizado éste.

3.3.1. Programa principal

*int main(int argc, char **argv)*: es el encargado de la ejecución de las funciones y métodos posteriormente descritos. El algoritmo del bucle principal del programa, encargado de la ejecución completa de un cliente, es el siguiente:

1º: Recibir el estado actual del servidor.
2º: Iniciar el motor de inteligencia artificial con la ontología y el cliente.
3º: Si no se ha terminado con la ejecución del programa:
- Cargar el estado actual en el motor de inteligencia artificial.
- Cargar la instancia del cliente en el motor de inteligencia artificial.
- Ejecutar el motor de inteligencia artificial para generar el archivo que guarda la acción.
- Enviar la acción al servidor mediante el archivo generado.
- Borrar el archivo que guarda la acción.
- Volver al paso 1º.

4º: Sino (se ha terminado con la ejecución del programa):
- Cargar el hecho de fin de traza en el motor de inteligencia artificial.
- Generar el archivo que guarda la acción de fin de ejecución.
- Enviar la acción al servidor mediante el archivo generado.
- Borrar el archivo que guarda la acción.

3.3.2. Funciones y métodos implementados

*buffer *GenerateActorKey(void)*: genera una clave aleatoria para el actor. Devuelve la clave en un buffer.

*void SendProfile(int ss, char *actor, char *stage, char *actor_id)*: envía el perfil del actor *actor* cuya clave es *actor_id* perteneciente al escenario *stage*, al servidor cuyo socket es *ss*.

*int ServerConnect(char *serverhost, int serverport, char *actor, char *stage)*: conecta el actor *actor* al servidor *serverhost* a través del puerto *serverport* en el escenario *stage*, mediante los protocolos de comunicación establecidos, asignándole una clave al actor y enviando el perfil del éste al servidor. Devuelve el socket del servidor.

*buffer *ReceiveState(int ss)*: recibe el estado actual de las entidades a través del socket del servidor *ss*. Devuelve el estado en un buffer.

*void SendAction(int ss, const char *f)*: envía la acción solicitada por el actor al socket del servidor *ss*. La acción está almacenada en el archivo *f*.

void killClient(int signal): captura la señal *signal* (*Control+C*) para terminar con la ejecución del cliente.

4. Ampliación del sistema

Para la creación del interfaz manual, ha sido necesario modificar la estructura de un cliente de inteligencia artificial.

Sólo se ha requerido modificar el archivo *client.c*. El archivo *client.clp* no ha sido modificado, debido a que es irrelevante cómo se organizan las decisiones del actor, ya que es el usuario el encargado de elegir la acción que quiere que ejecute el servidor.

4.1. Nuevos métodos implementados

4.1.1. void showActions(int num)

Muestra al usuario las acciones que puede realizar el actor en el turno actual. *num* contiene el número de acciones que puede realizar. El algoritmo utilizado se detalla a continuación:

1º: Obtener la primera activación de la agenda. La agenda contiene la secuencia de activaciones (acciones) ordenadas que puede realizar el actor en el turno actual.

2º: Mientras que i (iniciado a 0) sea menor que el número total de activaciones:

- Imprimir: i+1, el nombre de la activación y los objetos que utiliza ésta.
- Obtener la siguiente activación.
- Incrementar i.

4.1.2. void selectAction(void)

Permite que el usuario seleccione la acción que quiere que realice el actor en el turno actual. El algoritmo utilizado se detalla a continuación:

1º: Obtener la primera activación de la agenda.

2º: Mientras que haya más activaciones:

- Aumentar el contador de activaciones (iniciado a 0).
- Obtener la siguiente activación.

3º: Invocar al método *showActions*.

4º: Mientras que no haya errores en la introducción del número y éste no esté entre 0 y el número total de activaciones:

- Pedirle al usuario que introduzca el número de acción a ejecutar.
- Si hay algún error o el número introducido es 0:
  - Indicar que finalice la ejecución del programa

5º: Borrar todas las activaciones anteriores al número introducido, ya que la acción que se ejecuta es siempre la primera de la agenda.

4.2. Modificación del bucle principal del programa

Tras la implementación de los dos nuevos métodos que realizan la funcionalidad requerida, se ha de introducir la sentencia que permite al usuario seleccionar la acción a ejecutar, quedando el bucle principal del programa de la siguiente manera:

1º: Recibir el estado actual del servidor.
2º: Iniciar el motor de inteligencia artificial con la ontología y el cliente.
3º: Si no se ha terminado con la ejecución del programa:

- Cargar el estado actual en el motor de inteligencia artificial.
- Cargar la instancia del cliente en el motor de inteligencia artificial.
- *Seleccionar la acción a ejecutar (selectAction())*
- Ejecutar el motor de inteligencia artificial para generar el archivo que guarda la acción.
- Enviar la acción al servidor mediante el archivo generado.
- Borrar el archivo que guarda la acción.
- Volver al paso 1º.

4º: Sino (se ha terminado con la ejecución del programa):

- Cargar el hecho de fin de traza en el motor de inteligencia artificial.
- Generar el archivo que guarda la acción de fin de ejecución.
- Enviar la acción al servidor mediante el archivo generado.
- Borrar el archivo que guarda la acción.

5. Manual de usuario

A continuación se muestra el conjunto de pasos que hay que realizar para la instalación, configuración y utilización del juego AI-LIVE junto con la interfaz manual.

5.1. Requisitos generales

Se requiere una distribución de **Linux** (Debian, Ubuntu,…) para poder ejecutar la aplicación.

Se necesita también **PHP 4.3** o superior (http://www.php.net/downloads.php) con **CLI** para poder ejecutar los archivos *run* encargados de la ejecución del juego.
PHP (***P****HP* ***H****ypertext* ***P****re-processor*) es un lenguaje interpretado, multiplataforma, libre, y de propósito general, que está diseñado especialmente para desarrollo web, pero también puede emplearse para la creación de scripts.
CLI (***C****ommand* ***L****ine* ***I****nterface*) es una interfaz de programación de uso del servidor para **PHP**, que permite ejecutar aplicaciones o scripts implementadas en lenguaje **PHP** bajo línea de comandos.

No se requerirá ninguna versión de **CLIPS** (http://clipsrules.sourceforge.net/) para la ejecución de los archivos *.clp*, ya que se encuentra integrado en el lenguaje **C**.
CLIPS (***C L****anguage* ***I****ntegrated* ***P****roduction* ***S****ystem*) es una herramienta libre de programación, creada por la **NASA**, que permite realizar sistemas expertos basados en inteligencia artificial. Soporta programación lógica (para utilizar reglas de cumplimiento: Si A -> B), imperativa (para ejecutar algoritmos) y orientada a objetos (para diseñar sistemas complejos mediante módulos).

5.2. Configuración

La configuración del servidor y de los clientes está definida en los archivos *.ini*, incluidos en los directorios **AI-LIVE/server** y **AI-LIVE/client-xxx** respectivamente, donde **xxx** es el nombre del cliente:

- *server.ini*: en él se debe indicar dónde se encuentra la ontología (*ontology.clp*), el estado inicial (*initial.state*) y el puerto de escucha (*6969*).

- *client.ini*: en él se debe indicar dónde se encuentra la ontología (*ontology.clp*, la misma que en el servidor), el escenario inicial (*DEFAULT_STAGE*), el perfil del actor (*DEFAULT_ACTOR*), el servidor con el que conectarse (*localhost*) y el puerto de conexión a éste (*6969*).

El perfil de cada actor (*DEFAULT_ACTOR*) se encuentra en el directorio **AI-LIVE/client-xxx/profiles**. En él se deben indicar las características básicas (*name_*, *gender*, *weight*,…), otras relacionadas con el rol social (*age*, *sex*, *status*), y otras más específicas relacionadas con la personalidad (conscientiousness, *extraversion*, *neuroticism*, *openness*, *agreeableness*) y las emociones (*valence* y *arousal)* cuyo valor está comprendido entre -1 y 1, donde el valor 0 es la posición neutral.

5.3. Ejecución

5.3.1. Servidor y cliente manual

Antes de ejecutar el cliente manual, es necesario iniciar un terminal de comandos, accediendo desde éste al directorio del servidor **AI-LIVE/server** y ejecutar la sentencia *make* para generar el archivo ejecutable *server.* A continuación hay que ejecutar la sentencia *./run* .

Una vez hecho esto, se debe iniciar un nuevo terminal de comandos, accediendo ahora al directorio **AI-LIVE/client-manual** y ejecutar, de la misma forma, la sentencia *make* que generará el archivo ejecutable *client*, y a continuación la sentencia *./run* .

Por cada turno, se le mostrará al usuario el conjunto de acciones que puede realizar el actor en ese momento. Cada acción contiene un número que la identifica, su nombre, así como el conjunto de objetos que están involucrados en ella. Es por ello por lo que no se introducen parámetros para la ejecución de las acciones, ya que éstas contemplan todas las posibilidades que puede realizar el actor en cada turno.

El usuario debe introducir el número de la acción que desea que realice el actor, y pulsar *ENTER* para que posteriormente se envíe la acción al servidor y éste pueda ejecutarla.

Si el usuario desea salir de la aplicación, debe pulsar 0 y después *ENTER* en el terminal donde ha ejecutado el cliente manual, en cuyo momento se generará el archivo de traza (*traza.txt*). Seguidamente debe pulsar *Control+C* en el terminal donde ha ejecutado el servidor para finalizar.

5.3.2. Clientes de inteligencia artificial

También se pueden ejecutar clientes de inteligencia artificial junto con el servidor. Para ello es necesario iniciar un terminal de comandos, accediendo desde éste al directorio **AI-LIVE/client-xxx**, donde **xxx** es el nombre de un cliente distinto a **manual** y ejecutar la sentencia *./run* .

Para terminar la ejecución de estos clientes, basta con pulsar *Control+C* en el terminal donde se ha ejecutado el cliente de inteligencia artificial. Al igual que para el cliente manual, se generará el archivo de traza (*traza.txt*) para cada uno.

<u>Nota</u>: Si no permite ejecutar en un terminal de comandos la sentencia *./run* , ya sea en el servidor o en cualquiera de los clientes, asegurarse de que los archivos *run*, así como los archivos *client* y *server*, tienen permisos de ejecución como un programa.

6. Conclusiones

En este apartado se van a comentar las conclusiones a las que se ha llegado tras la implementación de la interfaz manual.

6.1. Conclusiones generales

La creación de la interfaz manual permite observar cómo influyen las acciones tomadas por el actor manual en las decisiones de los demás actores implementados con inteligencia artificial. Esto proporciona un sistema de prueba del comportamiento de este tipo de actores.

Además, garantiza una forma de comprobar que las acciones realizadas por los actores de inteligencia artificial están correctamente implementadas, ya que el cliente manual se basa estructuralmente en los clientes de inteligencia artificial.

El estudio de las decisiones tomadas por el cliente manual permite extrapolar la información de éste para la creación de nuevos clientes de inteligencia artificial, cuyo comportamiento se asemeje más al de un humano.

6.2. Conclusiones personales

La interfaz manual ha resultado más fácil de implementar gracias a que está basada principalmente en la estructura de un cliente de inteligencia artificial. Se han introducido nuevos métodos a los ya realizados para lograr el propósito del trabajo dirigido, evitando así desarrollar la aplicación desde cero.

La utilización del lenguaje **CLIPS** integrado en **C** ha permitido alcanzar el cometido del trabajo dirigido, introduciendo nuevas funcionalidades no usadas en **CLIPS**, como son la forma de impresión de la agenda en el formato indicado, así como la eliminación de la activación actual de ésta, que es aquella que se va a ejecutar.

7. Bibliografía

A continuación se muestra la bibliografía que ha servido de ayuda para la elaboración de la interfaz manual.

- CLIPS. Definición.
  http://es.wikipedia.org/wiki/CLIPS

- PHP. Definición.
  http://www.php-es.com

- CLIPS. Documentación.
  http://clipsrules.sourceforge.net/

- PHP. Documentación y descargas.
  http://www.php.net/

ANEXO A – Ejemplo de ejecución de AI-LIVE

Este anexo pretende mostrar al usuario un ejemplo de instalación, configuración y ejecución del juego AI-LIVE con la interfaz manual implementada.

Preinstalación

0. Comprobamos que tenemos instalado **PHP 4.3** o superior con **CLI**, accediendo al directorio **/etc/phpX/cli/**, donde **X** es la versión (4, 5,…). Si no se encuentra, instalaremos la última versión, abriendo un terminal de comandos, y ejecutando la sentencia *apt-get install php5 php5-cli* .

Instalación

1. Extraemos el archivo *ai_08_01_15.tar.gz* en el directorio **/home** de nuestra distribución **Linux**. En el nombre se indica la versión del juego, en donde *08* es el año, *01* el mes, y *15* es el día en el que se ha hecho la versión.

2. A continuación, extraemos el archivo *client-manual.tar.tgz* en el directorio **/home/ai_08_01_15/**, generado por la extracción del archivo anterior, creándose dentro un nuevo directorio llamado **/client-manual** que contiene el propio cliente manual.

Configuración

Servidor

3. Accedemos al directorio **/home/ai_08_01_15/server** y comprobamos que en el archivo *server.ini* la configuración por defecto es correcta, existiendo los archivos *ontology.clp* e *initial.state* en el directorio del servidor.

Cliente manual

4. Realizamos la misma operación para el cliente manual, accediendo al directorio **/home/ai_08_01_15/client-manual**. Una vez hecho esto, comprobamos que en el archivo *client.ini* la configuración por defecto es correcta, existiendo el archivo *ontology.clp* en el directorio del cliente manual, y que el puerto de conexión es el mismo que el de acceso al servidor, configurado anteriormente en el archivo *server.ini*.

5. No es necesario modificar el perfil del actor para el cliente manual, ya que vamos a ser nosotros los encargados de tomar las decisiones por él. Pero si se quiere modificar algunas características, el archivo *DEFAULT_ACTOR.profile* está situado en el directorio **/home/ai_08_01_15/client-manual/profiles**

Nota: Para más información sobre la configuración, consultar el apartado **5.2**.

Ejecución

6. Antes de continuar, comprobaremos en las propiedades, tanto del servidor como del cliente manual, que los archivos *run* tienen permisos de ejecución como programa, si no es el caso, asignarle esta característica.

Servidor

7. Abrimos un terminal de comandos, accedemos al directorio **/home/ai_08_01_15/server/** y ejecutamos la sentencia *make* , la cuál nos generará el archivo ejecutable *server*, y seguidamente la sentencia *./run* , mostrándose esto:

```
Defining defglobal: numVC
Defining deffunction: sign
Defining defgeneric: printExp
   Method #1 defined.
Defining defrule: Gossip +j+j+j+j+j
Defining defrule: TellLikenessShared =j+j+j+j+j+j+j
Defining defrule: TellLikenessNotShared =j=j=j+j
Defining defrule: createRelationship +j+j
Defining defrule: createExplicitRelationship +j
Defining defrule: modifyRelationship +j+j
Defining defrule: createLikeness +j
*****************************************************************
*********  CLIPS: Initialised... State files written  **********
*****************************************************************
server: Listener enabled
```

Cliente manual

8. Abrimos un nuevo terminal de comandos, accedemos al directorio **/home/ai_08_01_15/client-manual/** y ejecutamos la sentencia *make* para generar el archivo ejecutable *client*, y a continuación la sentencia *./run* , mostrándose esto:

```
./client: Connected, receiving first state

ACTIONS:

1 - assigningBag:
[CLIENTID],[ASSIGNINGBAG],[DEFAULT_STAGE],[CLIPS_ACTOR_VVQGGEM9RT],[Ba
g_3_id]
2 - assigningBag:
[CLIENTID],[ASSIGNINGBAG],[DEFAULT_STAGE],[CLIPS_ACTOR_VVQGGEM9RT],[Ba
g_2_id]
3 - assigningBag:
[CLIENTID],[ASSIGNINGBAG],[DEFAULT_STAGE],[CLIPS_ACTOR_VVQGGEM9RT],[Ba
g_1_id]
4 - createLikeness: [CREATE],[CLIENTID],
5 - workObjectGoal:
[MOVEACTION],[OURGOAL],[DEFAULT_STAGE],[Computer_1_id],[CLIENTID],[CLI
PS_ACTOR_VVQGGEM9RT],[INSTANCE_010]
6 - FindFreeCell:
[FINDFREECELL],[DEFAULT_STAGE],[CLIENTID],[CLIPS_ACTOR_VVQGGEM9RT],[IN
STANCE_098]
7 - FindFreeCell:
[FINDFREECELL],[DEFAULT_STAGE],[CLIENTID],[CLIPS_ACTOR_VVQGGEM9RT],[IN
STANCE_097]
8 - FindFreeCell:
[FINDFREECELL],[DEFAULT_STAGE],[CLIENTID],[CLIPS_ACTOR_VVQGGEM9RT],[IN
STANCE_096]
...
96 - FindFreeCell:
[FINDFREECELL],[DEFAULT_STAGE],[CLIENTID],[CLIPS_ACTOR_VVQGGEM9RT],[IN
STANCE_002]
97 - FindFreeCell:
[FINDFREECELL],[DEFAULT_STAGE],[CLIENTID],[CLIPS_ACTOR_VVQGGEM9RT],[IN
STANCE_001]
98 - FindFreeCell:
[FINDFREECELL],[DEFAULT_STAGE],[CLIENTID],[CLIPS_ACTOR_VVQGGEM9RT],[IN
STANCE_000]

Insert the action's number to run (0 = EXIT) and press ENTER:
```

9. Ahora elegimos una acción para que la ejecute el servidor, por ejemplo la tercera, para asignarle al actor la bolsa Bag_1_id. Para ello introduciremos *3* y pulsaremos *ENTER*, mostrándose esto:

```
---------------------------------- RUN
FIN RUN ----------------------------------

./client: Action sent
./client: Receiving state.

********** regla AssigningBag **********

     - bolsa asignada : [Bag_1_id]

ACTIONS:

1 - createLikeness: [CREATE],[CLIENTID],
2 - workObjectGoal:
[MOVEACTION],[OURGOAL],[DEFAULT_STAGE],[Computer_1_id],[CLIENTID],[CLI
PS_ACTOR_VVQGGEM9RT],[INSTANCE_010]
3 - FindFreeCell:
[FINDFREECELL],[DEFAULT_STAGE],[CLIENTID],[CLIPS_ACTOR_VVQGGEM9RT],[IN
STANCE_098]
4 - FindFreeCell:
[FINDFREECELL],[DEFAULT_STAGE],[CLIENTID],[CLIPS_ACTOR_VVQGGEM9RT],[IN
STANCE_097]
5 - FindFreeCell:
[FINDFREECELL],[DEFAULT_STAGE],[CLIENTID],[CLIPS_ACTOR_VVQGGEM9RT],[IN
STANCE_096]
...
93 - FindFreeCell:
[FINDFREECELL],[DEFAULT_STAGE],[CLIENTID],[CLIPS_ACTOR_VVQGGEM9RT],[IN
STANCE_002]
94 - FindFreeCell:
[FINDFREECELL],[DEFAULT_STAGE],[CLIENTID],[CLIPS_ACTOR_VVQGGEM9RT],[IN
STANCE_001]
95 - FindFreeCell:
[FINDFREECELL],[DEFAULT_STAGE],[CLIENTID],[CLIPS_ACTOR_VVQGGEM9RT],[IN
STANCE_000]

Insert the action's number to run (0 = EXIT) and press ENTER:
```

10. A continuación, elegimos la primera acción, para crear los gustos del cliente. Para ello introduciremos *1* y pulsaremos *ENTER*, mostrándose esto:

```
----------------------------------- RUN
FIN RUN -----------------------------------

./client: Action sent
./client: Receiving state.

Create likeness sent[CLIPS_ACTOR_VVQGGEM9RT]

ACTIONS:

1 - workObjectGoal:
[MOVEACTION],[OURGOAL],[DEFAULT_STAGE],[Computer_1_id],[CLIENTID],[CLI
PS_ACTOR_VVQGGEM9RT],[INSTANCE_010]
2 - FindFreeCell:
[FINDFREECELL],[DEFAULT_STAGE],[CLIENTID],[CLIPS_ACTOR_VVQGGEM9RT],[IN
STANCE_098]
3 - FindFreeCell:
[FINDFREECELL],[DEFAULT_STAGE],[CLIENTID],[CLIPS_ACTOR_VVQGGEM9RT],[IN
STANCE_097]
4 - FindFreeCell:
[FINDFREECELL],[DEFAULT_STAGE],[CLIENTID],[CLIPS_ACTOR_VVQGGEM9RT],[IN
STANCE_096]
...
92 - FindFreeCell:
[FINDFREECELL],[DEFAULT_STAGE],[CLIENTID],[CLIPS_ACTOR_VVQGGEM9RT],[IN
STANCE_002]
93 - FindFreeCell:
[FINDFREECELL],[DEFAULT_STAGE],[CLIENTID],[CLIPS_ACTOR_VVQGGEM9RT],[IN
STANCE_001]
94 - FindFreeCell:
[FINDFREECELL],[DEFAULT_STAGE],[CLIENTID],[CLIPS_ACTOR_VVQGGEM9RT],[IN
STANCE_000]

Insert the action's number to run (0 = EXIT) and press ENTER:
```

11. Ahora, elegimos la primera acción, para acceder a un objeto de trabajo. Para ello introduciremos *1* y pulsaremos *ENTER*, mostrándose esto:

```
---------------------------------- RUN
FIN RUN ----------------------------------

./client: Action sent
./client: Receiving state.

********** regla WorkObjectGoal **********

     - vamos a por: [Computer_1_id]
     - situado en x: 1 y: 1
     - coordenadas actuales del actor: X: 2 Y:1
     - celda de acceso : [INSTANCE_010]

********** regla move **********

     - Se va a mover el actor con el Id: [CLIPS_ACTOR_VVQGGEM9RT]
     - Coordenadas futuras del actor: X= 1 Y= 0

ACTIONS:

1 - Working:
[CLIENTID],[WORKING],[DEFAULT_STAGE],[CLIPS_ACTOR_VVQGGEM9RT],[Compute
r_1_id],[INSTANCE_010]
2 - FindFreeCell:
[FINDFREECELL],[DEFAULT_STAGE],[CLIENTID],[CLIPS_ACTOR_VVQGGEM9RT],[IN
STANCE_098]
3 - FindFreeCell:
[FINDFREECELL],[DEFAULT_STAGE],[CLIENTID],[CLIPS_ACTOR_VVQGGEM9RT],[IN
STANCE_097]
4 - FindFreeCell:
[FINDFREECELL],[DEFAULT_STAGE],[CLIENTID],[CLIPS_ACTOR_VVQGGEM9RT],[IN
STANCE_096]
...
92 - FindFreeCell:
[FINDFREECELL],[DEFAULT_STAGE],[CLIENTID],[CLIPS_ACTOR_VVQGGEM9RT],[IN
STANCE_002]
93 - FindFreeCell:
[FINDFREECELL],[DEFAULT_STAGE],[CLIENTID],[CLIPS_ACTOR_VVQGGEM9RT],[IN
STANCE_001]
94 - FindFreeCell:
[FINDFREECELL],[DEFAULT_STAGE],[CLIENTID],[CLIPS_ACTOR_VVQGGEM9RT],[IN
STANCE_000]

Insert the action's number to run (0 = EXIT) and press ENTER:
```

Como se puede observar, el servidor ha realizado dos acciones, una para seleccionar el objeto de trabajo (*WorkObjectGoal*) y otra para moverse a la casilla de acceso a éste (*move*), sin embargo en los pasos anteriores sólo realizaba una.

Esto es debido a que las acciones *xxxObjectGoal* implican que, una vez se ha seleccionado el objeto, se mueva el actor mediante la acción de movimiento *move*, lo que no ocurre con el resto de las acciones.

12. Posteriormente seleccionamos la primera acción, para que el actor utilice el objeto de trabajo. Para ello introduciremos *1* y pulsaremos *ENTER*, mostrándose esto:

```
---------------------------------- RUN
FIN RUN ----------------------------------

./client: Action sent
./client: Receiving state.

********** regla Working **********

     - objeto de trabajo : [Computer_1_id]

ACTIONS:

1 - read:
[CLIENTID],[READ],[DEFAULT_STAGE],[CLIPS_ACTOR_VVQGGEM9RT],[Bag_1_id],
[Book_1_id]
2 - play:
[CLIENTID],[PLAY],[DEFAULT_STAGE],[CLIPS_ACTOR_VVQGGEM9RT],[Bag_1_id],
[Ball_1_id]
3 - FindFreeCell:
[FINDFREECELL],[DEFAULT_STAGE],[CLIENTID],[CLIPS_ACTOR_VVQGGEM9RT],[IN
STANCE_098]
4 - FindFreeCell:
[FINDFREECELL],[DEFAULT_STAGE],[CLIENTID],[CLIPS_ACTOR_VVQGGEM9RT],[IN
STANCE_097]
5 - FindFreeCell:
[FINDFREECELL],[DEFAULT_STAGE],[CLIENTID],[CLIPS_ACTOR_VVQGGEM9RT],[IN
STANCE_096]
...
93 - FindFreeCell:
[FINDFREECELL],[DEFAULT_STAGE],[CLIENTID],[CLIPS_ACTOR_VVQGGEM9RT],[IN
STANCE_002]
94 - FindFreeCell:
[FINDFREECELL],[DEFAULT_STAGE],[CLIENTID],[CLIPS_ACTOR_VVQGGEM9RT],[IN
STANCE_001]
95 - FindFreeCell:
[FINDFREECELL],[DEFAULT_STAGE],[CLIENTID],[CLIPS_ACTOR_VVQGGEM9RT],[IN
STANCE_000]

Insert the action's number to run (0 = EXIT) and press ENTER:
```

Fin de ejecución

Cliente manual

13. Para finalizar con la ejecución del cliente manual, introduciremos *0* y pulsaremos *ENTER* en el terminal de comandos donde se ha ejecutado el cliente manual, mostrándose esto:

```
----------------------------------- RUN

********** regla read **********

     - objecto de lectura  seleccionado : [Book_1_id]

FIN RUN -----------------------------------

./client: Action sent
./client: Receiving state.

Client TERMINATED
```

Al terminar con la ejecución, se enviará al servidor la última acción, que será la primera de la lista de acciones anteriormente mostrada (*read*).

Una vez hecho esto, se generará el archivo *traza.txt* en el directorio del cliente manual.

Servidor

14. Para finalizar con la ejecución del servidor, pulsaremos *Control+C* en el terminal de comandos donde se ha ejecutado el servidor, mostrándose esto:

```
...
Playing CLIPS_ACTOR_VVQGGEM9RT on stage DEFAULT_STAGE
************* - Action: Exit_Actor **********
******* Actor: [CLIPS_ACTOR_VVQGGEM9RT] goes out of stage
[DEFAULT_STAGE]
```